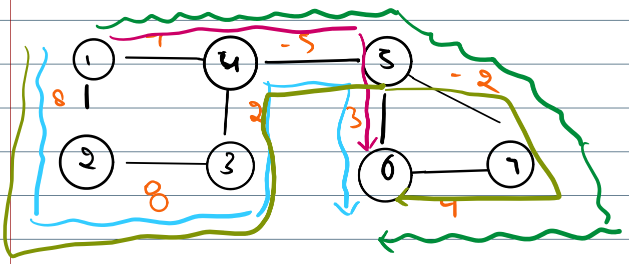
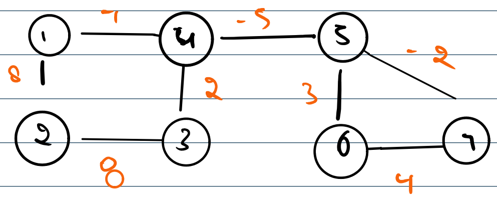
Graphs

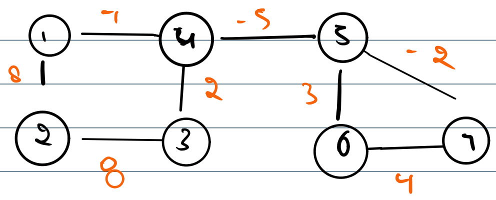


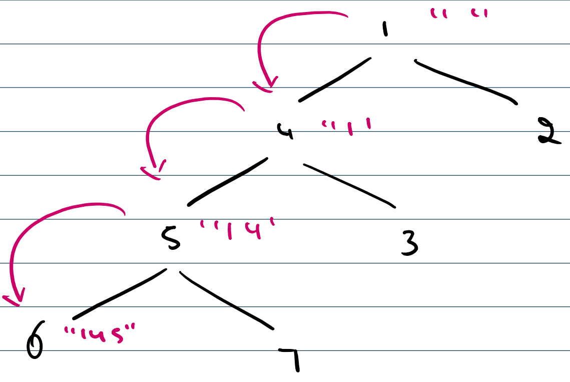


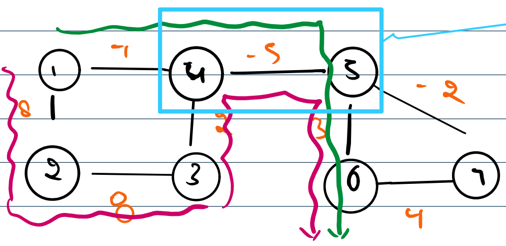
Find all paths

source → 1
target → 6

1 → 4 → 5 → 7 → 6
1 → 2 → 3 → 4 → 5 → 6
1 → 4 → 5 → 6
1 → 2 → 3 → 4 → 5 → 7 → 6

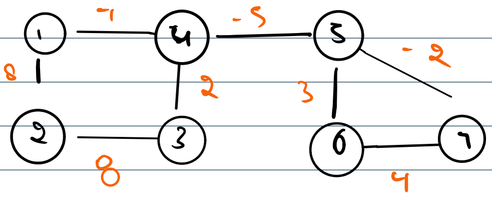


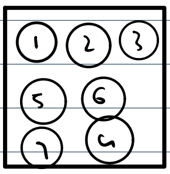


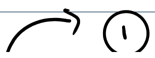


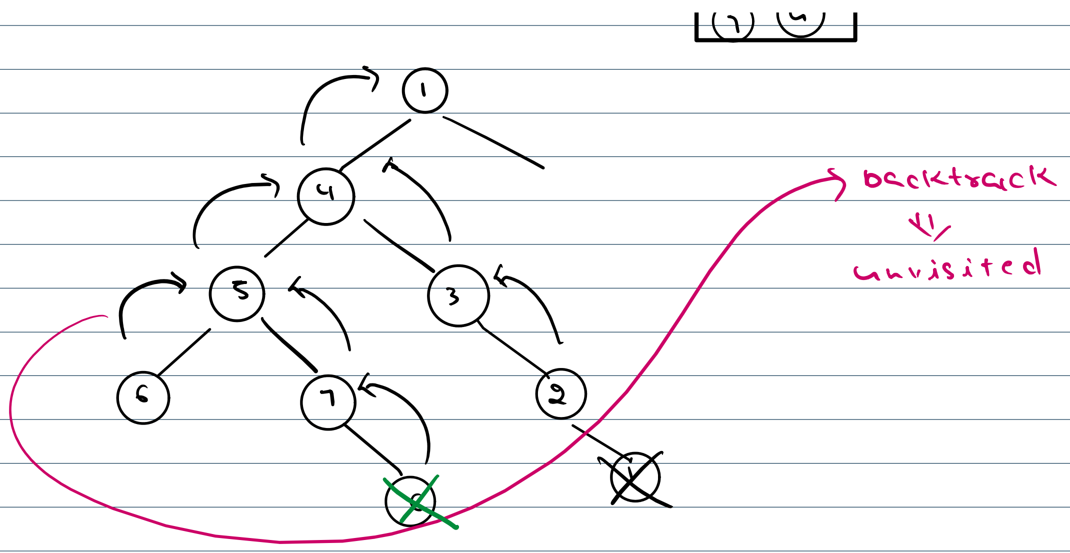
intersection
⇓
4 → 5 ⇒ visited ⇒ Pink

Green ⇒ 4 → 5 ⇒ ~~al~~ already visited





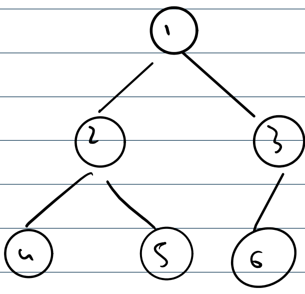




To Solve Graph problems

1) - DFS (Depth First search)
2) - BFS (Breadth First Search)

BFS



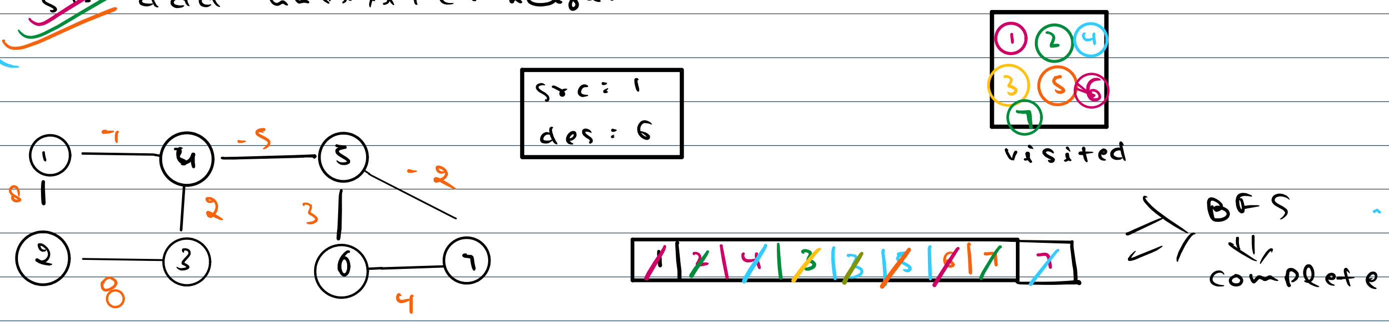
1 2 3 4 5 6 ⇒ Level order Traversal

DFS

1 2 4 5 3 6

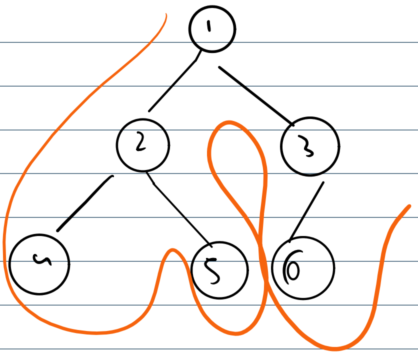
5 steps for every graph question

1) - Remove
2) - Ignore → already in visited ⇒ don't do steps 3,4,5
3) - mark as visited
4) - self work
5) - add unvisited neigh.



DFS → Depth First Search

1 2 4 5 3 6



$\Rightarrow$ Data structure

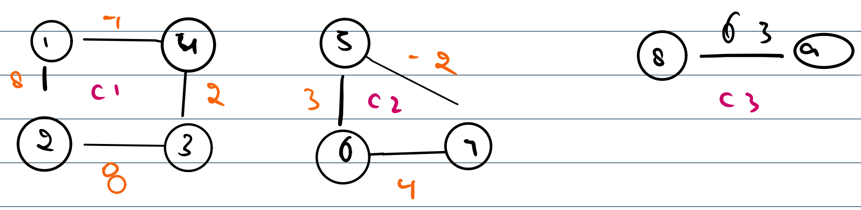
$\Downarrow$

Recursion

$\Downarrow$

Stack

Disconnected graphs



1. Count components.
2. Check if the graph is connected.
3. Work with disconnected graphs.
4. Traversal
   - $\longrightarrow$ BFS $\longrightarrow$ BFT
   - $\longrightarrow$ DFS $\longrightarrow$ DFT

```
public boolean bfs(int src, int des) {
        Queue<Integer> q = new LinkedList<>();
        HashSet<Integer> visited = new HashSet<>();
        q.add(src);

        while (!q.isEmpty()) {
//          1. remove
            int r = q.poll();

//          2. ignore if already visited
            if(visited.contains(r)) {
                continue;
            }

//          3. marked visited
            visited.add(r);

//          4. self work
            if(r == des) {
```

$\rightarrow$ Repeat it all vertices

$\Downarrow$

put inside a loop

```
//        4. self work
           if(r == des) {
               return true;
           }

//        5. add unvisited neighbours
           for(int nbrs: map.get(r).keySet()) {
               if(!visited.contains(nbrs)) {
                   q.add(nbrs);
               }
           }
       }
       return false;
   }
```